Audio Session Programming Guide

Developer

Contents

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

Figures, Tables, and Listings

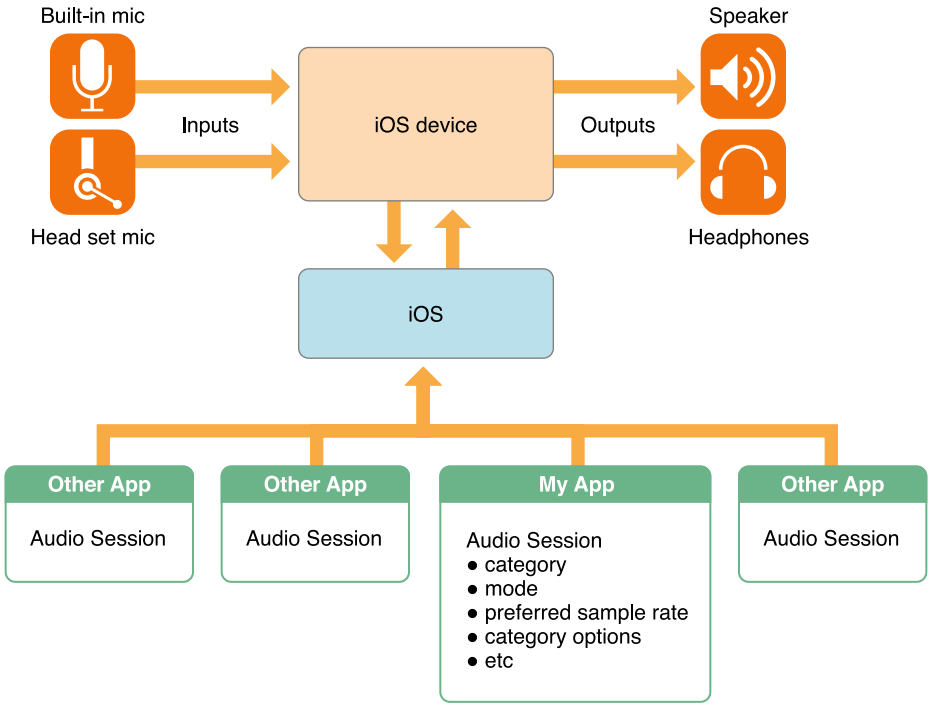
2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

SwiftObjective-C

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

Introduction

iOS handles audio behavior at the app, inter-app, and device levels through *audio sessions* and the `AVAudioSession` APIs.



Using the `AVAudioSession` API, you resolve questions such as:

- Should your app's audio be silenced by the Ring/Silent switch? Yes, if audio is not essential to your app. An example is an app that lets users take notes in a meeting without its sound disturbing others. But a pronouncing dictionary should ignore the Ring/Silent switch and always play—the central purpose of the app requires sound.
- Should music continue when your audio starts? Yes, if your app is a virtual piano that lets users play along to songs from their music libraries. On the other hand, music should stop when your app starts if your app plays streaming Internet radio.

Users may plug in or unplug headsets, phone calls may arrive, and alarms may sound. Indeed, the audio environment on an iOS device is quite complex. iOS does the heavy lifting, while you employ audio session APIs to specify configuration and to respond gracefully to system requests, using very little code.

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

At a Glance

`AVAudioSession` gives you control your app's audio behavior. You can:

- Select the appropriate input and output routes for your app
- Determine how your app integrates audio from other apps
- Handle interruptions from other apps
- Automatically configure audio for the type of app your are creating

An Audio Session Encapsulates a Set of Behaviors

An *audio session* is the intermediary between your app and iOS used to configure your app's audio behavior. Upon launch, your app automatically gets a singleton audio session.

Relevant Chapters: Defining an Audio Session (page 10).

Categories Express Audio Roles

The primary mechanism for expressing audio behaviors is the audio session category. By setting the category, you indicate whether your app uses input or output routes, whether you want music to continue playing along with your audio, and so on. The behavior you specify should meet user expectations as described in Sound in *iOS Human Interface Guidelines* .

Seven audio session categories, along with a set of override and modifier switches, let you customize audio behavior according to your app's personality or role. Various categories support playback, recording, and playback along with recording. When the system knows your app's audio role, it affords you appropriate access to hardware resources. The system also ensures that other audio on the device behaves in a way that works for your app; for example, if you need the Music app to be interrupted, it is.

Relevant Chapters: Working with Categories (page 18) and Responding to Interruptions (page 26).

Modes Customize Categories

Users expect certain behaviors from certain categories of apps. Modes specialize the behavior of a given category. For example, when an app uses the Video Recording mode, the system may choose a different built-in microphone than it would if it was using the default mode. The system may also engage microphone signal processing that is tuned for video recording use cases.

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

Relevant Chapters: Working with Categories (page 18).

Notifications Support Interruption Handling

An *audio interruption* is the deactivation of your app's audio session—which immediately stops your audio. Interruptions occur when a competing audio session from an app activates and that session is not categorized by the system to mix with yours. After your session goes inactive, the system sends a "you were interrupted" message which you can respond to by saving state, updating the user interface, and so on.

To handle interruptions, register for `AVAudioSessionInterruptionNotification` provided in `AVAudioSession`. Write your `beginInterruption` and `endInterruption` methods to ensure the minimum possible disruption, and the most graceful possible recovery, from the perspective of the user.

Relevant Chapters: Responding to Interruptions (page 26).

Notifications Support Audio Route Change Handling

Users have particular expectations when they initiate an *audio route change* by docking or undocking a device, or by plugging in or unplugging a headset. Sound in *iOS Human Interface Guidelines* describes these expectations and provides guidelines on how to meet them. Handle route changes by registering for `AVAudioSessionRouteChangeNotification`.

Relevant Chapters: Responding to Route Changes (page 37).

Categories Support Advanced Features

You can fine-tune an audio session category in a variety of ways. Depending on the category, you can:

- Allow other audio (such as from the Music app) to mix with yours when a category normally disallows it.
- Change the audio output route from the receiver to the speaker.
- Allow Bluetooth audio input.
- Specify that other audio should reduce in volume ("duck") when your audio plays.
- Optimize your app for device hardware at runtime. Your code adapts to the device it's running on and to changes initiated by the user (such as by plugging in a headset) as your app runs.

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

Relevant Chapters: Working with Categories (page 18), Optimizing Your App for Device Hardware (page 32), and Fine-Tuning an Audio Session for Players (page 40)

Prerequisites

Be familiar with Cocoa Touch development as introduced in *App Programming Guide for iOS* and with the basics of Core Audio as described in that document and in *Core Audio Overview*. Because audio sessions bear on practical end-user scenarios also be familiar with iOS devices, and with iOS Human Interface Guidelines, especially the Sound section in *iOS Human Interface Guidelines*.

See Also

You may find the following resources helpful:

- *AVAudioSession Class Reference*, which describes the Objective-C interface for configuring and using this technology.
- *AddMusic*, a sample code project that demonstrates use of an audio session object in the context of a playback app. This sample also demonstrates coordination between app audio and Music app audio.

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

Defining an Audio Session

Objective-CSwift

An *audio session* is the intermediary between your app and iOS for configuring audio behavior. Upon launch, your app automatically gets a singleton audio session. You configure it to express your app's audio intentions. For example:

- Will you mix your app's sounds with those from other apps (such as the Music app), or do you intend to silence other audio?
- How should your app to respond to an audio interruption, such as a Clock alarm?
- How should your app respond when a user plugs in or unplugs a headset?

Audio session configuration influences all audio activity while your app is running, except for user-interface sound effects played through the System Sounds Services API. You can query the audio session to discover hardware characteristics of the device your app is on, such as channel count and sample rate. These characteristics can vary by device and can change according to user actions while your app runs.

You can explicitly activate and deactivate your audio session. For app sound to play, or for recording to work, your audio session must be active. The system can also deactivate your audio session—which it does, for example, when a phone call arrives or an alarm sounds. Such a deactivation is called an *interruption*. The audio session APIs provide ways to respond to and recover from interruptions.

Audio Session Default Behavior

An audio session comes with some default behavior. Specifically:

- Playback is enabled, and recording is disabled.
- When the user moves the Silent switch (or Ring/Silent switch on iPhone) to the "silent" position, your audio is silenced.
- When the user presses the Sleep/Wake button to lock the screen, or when the Auto-Lock period expires, your audio is silenced.
- When your audio starts, other audio on the device—such as Music app audio that was already playing—is silenced.

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

All of the above behavior is provided by the default audio session category, `AVAudioSessionCategorySoloAmbient`. Working with Categories (page 18) provides information on how to incorporate a category into your app.

Your audio session begins automatically when you start to play or record audio; however, relying on the default activation is a risky state for your app. For example, if an iPhone rings and the user ignores the call—leaving your app running—your audio may no longer play, depending on which playback technology you're using. The next section describes some strategies for dealing with such issues, and Responding to Interruptions (page 26) goes into depth on the topic.

You can take advantage of default behavior as you're bringing your app to life during development. However, the only times you can safely ignore the audio session for a shipping app are these:

- Your app uses System Sound Services or the UIKit `playInputClick` method for audio and uses no other audio APIs.

  System Sound Services is the iOS technology for playing user-interface sound effects and for invoking vibration. It is unsuitable for other purposes. (See *System Sound Services Reference* and the *Audio UI Sounds (SysSound)* sample code project.)

  The UIKit `playInputClick` method lets you play standard keyboard clicks in a custom input or keyboard accessory view. Its audio session behavior is handled automatically by the system. See Playing Input Clicks in *Text Programming Guide for iOS*.

- Your app uses no audio at all.

In all other cases, do not ship your app with the default audio session.

Why a Default Audio Session Usually Isn't What You Want

If you don't initialize, configure, and explicitly use your audio session, your app cannot respond to interruptions or audio hardware route changes. Moreover, your app has no control over OS decisions about audio mixing between apps.

Here are some scenarios that demonstrate audio session default behavior and how you can change it:

- Scenario 1. You write an audio book app. A user begins listening to *The Merchant of Venice*. As soon as Lord Bassanio enters, Auto-Lock times out, the screen locks, and your audio goes silent.

  To ensure that audio continues upon screen locking, configure your audio session to use a category that supports playback and set the audio flag in `UIBackgroundModes`.

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

- Scenario 2. You write a first-person shooter game that uses OpenAL-based sound effects. You also provide a background soundtrack but include an option for the user to turn it off and play a song from the user's music library instead. After starting up a motivating song, the user fires a photon torpedo at an enemy ship, and the music stops.

  To ensure that music is not interrupted, configure your audio session to allow mixing. Use the `AVAudioSessionCategoryAmbient` category, or modify the `AVAudioSessionCategoryPlayback` category to support mixing.

- Scenario 3. You write a streaming radio app that uses Audio Queue Services for playback. While a user is listening, a phone call arrives and stops your sound, as expected. The user chooses to ignore the call and dismisses the alert. The user taps Play again to resume the music stream, but nothing happens. To resume playback, the user must quit your app and restart it.

  To handle the interruption of an audio queue gracefully, set the appropriate category and register for `AVAudioSessionInterruptionNotification` and have your app respond accordingly.

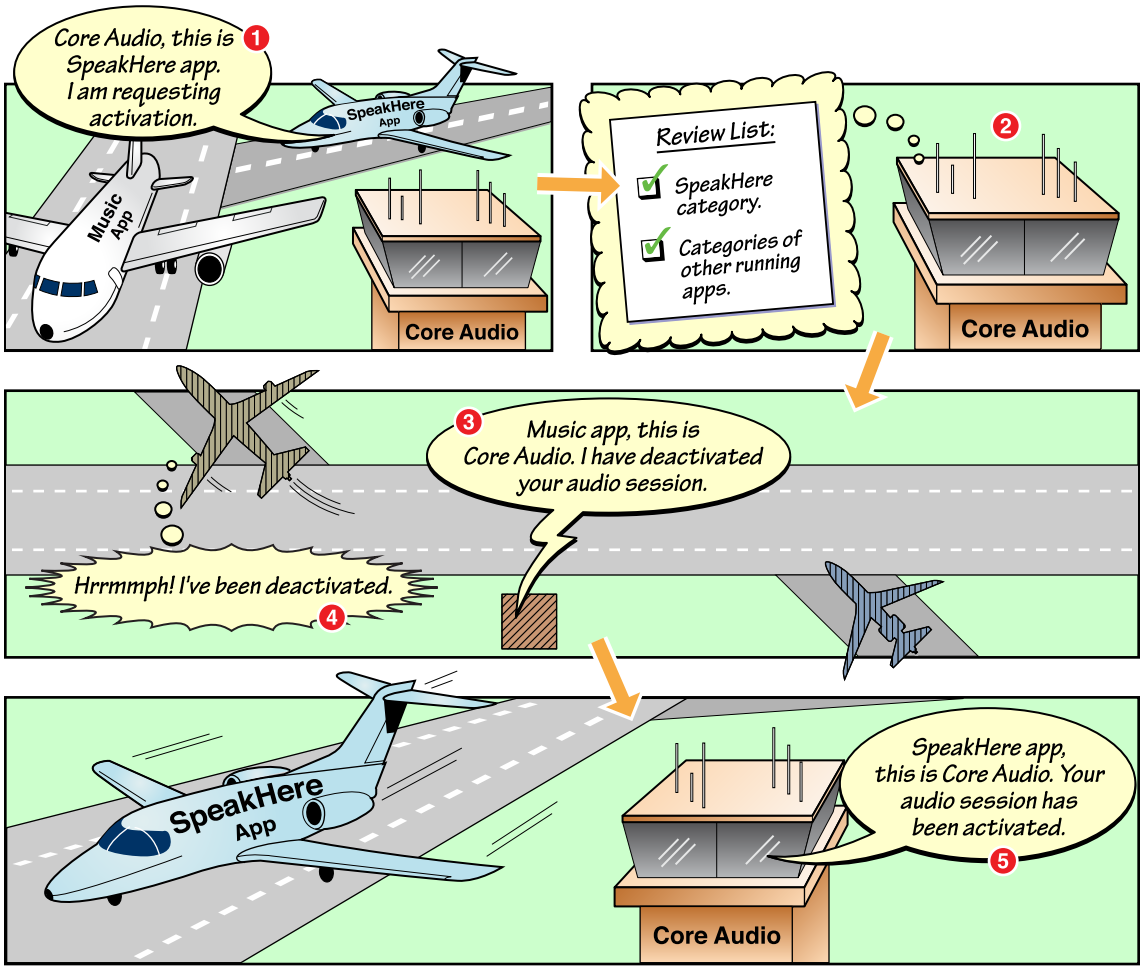
How the System Resolves Competing Audio Demands

As your iOS app launches, built-in apps (Messages, Music, Safari, the phone) may be running in the background. Each of these may produce audio: a text message arrives, a podcast you started 10 minutes ago continues playing, and so on.

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

If you think of an iOS device as an airport, with apps represented as taxiing planes, the system serves as a sort of control tower. Your app can make audio requests and state its desired priority, but final authority over what happens "on the tarmac" comes from the system. You communicate with the "control tower" using the audio session. Figure 1-1 illustrates a typical scenario—your app wanting to use audio while the Music app is already playing. In this scenario your app interrupts the Music app.

Figure 1-1 The system manages competing audio demands



In step 1 of the figure, your app requests activation of its audio session. You'd make such a request, for example, on app launch, or perhaps in response to a user tapping the Play button in an audio recording and playback app. In step 2, the system considers the activation request. Specifically, it considers the category you've assigned to your audio session. In Figure 1-1, the SpeakHere app uses a category that requires other audio to be silenced.

In steps 3 and 4, the system deactivates the Music app's audio session, stopping its audio playback. Finally, in step 5, the system activates the SpeakHere app's audio session and playback can begin.

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

The system has final authority to activate or deactivate any audio session present on a device. The system follows the inviolable rule that "the phone always wins." No app, no matter how vehemently it demands priority, can trump the phone. When a call arrives, the user gets notified and your app is interrupted—no matter what audio operation you have in progress and no matter what category you have set.

Incorporating an AVCaptureSession

The AV Foundation capture APIs (`AVCaptureDevice`, `AVCaptureSession`) enable you to capture synchronized audio and video from camera and microphone inputs. As of iOS 7, the `AVCaptureDevice` object that represents microphone input can share your app's `AVAudioSession`. By default, `AVCaptureSession` will configure your `AVAudioSession` optimally for recording when your `AVCaptureSession` is using the microphone. Set the `automaticallyConfiguresApplicationAudioSession` property to NO to override the default behavior and the `AVCaptureDevice` will use your current `AVAudioSession` settings without altering them. See *AVCaptureSession Class Reference* and the Media Capture chapter for more information.

Initializing Your Audio Session

The system provides an audio session object upon launch of your app. However, you must initialize the session in order to handle interruptions.

The AV Foundation framework for managing interruptions takes advantage of the implicit initialization that occurs when you obtain a reference to the `AVAudioSession` object, as shown here:

```
// implicitly initializes your audio session
AVAudioSession *session = [AVAudioSession sharedInstance];
```

The `session` variable now represents the initialized audio session and can be used immediately. Apple recommends implicit initialization when you handle audio interruptions with the `AVAudioSession` class's interruption notifications, or with the delegate protocols of the `AVAudioPlayer` and `AVAudioRecorder` classes.

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

Adding Volume and Route Control

Use the `MPVolumeView` class to present volume and routing control for your app. The volume view provides a slider to control the volume from inside your app and a button for choosing the output audio route. Apple recommends using the `MPVolumeView` route picker over `AVAudioSessionPortOverride` when routing audio to the built-in speaker. See *MPVolumeView Class Reference*.

Responding to Remote Control Events

Remote control events let users control an app's multimedia. If your app plays audio or video content, you might want it to respond to remote control events that originate from either transport controls or external accessories. iOS converts commands in `UIEvent` objects and delivers the events to the app. The app sends them to the first responder and, if the first responder doesn't handle them, they travel up the responder chain.

Your app must be the "Now Playing" app and currently playing audio before it can respond to events. See Remote Control Events and *MPNowPlayingInfoCenter Class Reference*.

Activating and Deactivating Your Audio Session

The system activates your audio session on app launch. Even so, Apple recommends that you explicitly activate your session—typically as part of your app's `viewDidLoad` method, and set preferred hardware values prior to activating your audio session. See Setting Preferred Hardware Values (page 33) for a code example. This gives you an opportunity to test whether activation succeeded. However, if your app has a play/pause UI element, write your code so that the user must press play before the session is activated. Likewise, when changing your audio session's active/inactive state, check to ensure that the call is successful. Write your code to gracefully handle the system's refusal to activate your session.

The system deactivates your audio session for a Clock or Calendar alarm or incoming phone call. When the user dismisses the alarm, or chooses to ignore a phone call, the system allows your session to become active again. Whether to reactivate a session at the end of an interruption depends on the app type, as described in Audio Guidelines By App Type (page 43).

Listing 1-1 shows how to activate your audio session.

Listing 1-1 Activating an audio session using the AV Foundation framework

```
NSError *activationError = nil;

BOOL success = [[AVAudioSession sharedInstance] setActive: YES error:
&activationError];
```

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

```
if (!success) { /* handle the error in activationError */ }
```

To deactivate your audio session, pass `NO` to the `setActive` parameter.

In the specific case of playing or recording audio with an `AVAudioPlayer` or `AVAudioRecorder` object, the system takes care of audio session reactivation upon interruption end. Nonetheless, Apple recommends that you register for notification messages and explicitly reactivate your audio session. Thus you can ensure that reactivation succeeded, and you can update your app's state and user interface. For more on this, see Figure 3-1 (page 28).

Most apps never need to deactivate their audio session explicitly. Important exceptions include VoIP (Voice over Internet Protocol) apps, turn-by-turn navigation apps, and, in some cases, recording apps.

- Ensure that the audio session for a VoIP app, which usually runs in the background, is active only while the app is handling a call. In the background, standing ready to receive a call, a VoIP app's audio session should not be active.
- Ensure that the audio session for an app using a recording category is active only while recording. Before recording starts and when recording stops, ensure your session is inactive to allow other sounds, such as incoming message alerts, to play.

Checking Whether Other Audio Is Playing During App Launch

When a user launches your app, sound may already be playing on the device. For example, the Music app may be playing music, or Safari may be streaming audio when your app launches. This situation is especially salient if your app is a game. Many games have a music sound track as well as sound effects. In this case, Sound in *iOS Human Interface Guidelines* advises you to assume that users expect the other audio to continue as they play your game while still playing the game's sound effects.

Inspect the `otherAudioPlaying` property to determine if audio is already playing when you launch your app. If other audio is playing when your app launches, mute your game's soundtrack and assign the `AVAudioSessionCategorySoloAmbient` category. See Working with Categories (page 18) for more information on categories.

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

Working with Inter-App Audio

In its most basic form, inter-app audio allows one app, called the node app, to send its audio output to another app, called the host app. However, it is also possible for the host app to send its output to the node app, have the node app process the audio, and send the result back to the host app. The host app needs to have an active audio session; however, the node app only needs an active audio session if it is receiving input from the host app or the system. Use the following guidelines when setting up inter-app audio:

- Set the “inter-app-audio” entitlement for both the host and node app.
- Set the UIBackgroundModes audio flag for the host app.
- Set the UIBackgroundModes audio flag for node apps that use audio input or output routes while simultaneously connected to an inter-app audio host.
- Set the `AVAudioSessionCategoryOptionMixWithOthers` for both the host and node app.
- Make sure the node app’s audio session is active if it receives audio input from the system (or sends audio output) while simultaneously connected to an inter-app host.

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

Working with Categories

Objective-CSwift

An audio session *category* is a key that identifies a set of audio behaviors for your app. By setting a category, you indicate your audio intentions to the system—such as whether your audio should continue when the Ringer/Silent switch is flipped. The seven audio session categories in iOS, along with a set of override and modifier switches, let you customize your app's audio behavior.

Each audio session category specifies a particular pattern of "yes" and "no" for each of the following behaviors, as detailed in Table B-1 (page 47):

- *Interrupts non-mixable apps audio*: If yes, non-mixable apps will be interrupted when your app activates its audio session.
- *Silenced by the Silent switch*: If yes, your audio is silenced when the user moves the Silent switch to silent. (On iPhone, this switch is called the *Ring/Silent switch* .)
- *Supports audio input*: If yes, app audio input (recording), is allowed.
- *Supports audio output*: If yes, app audio output (playback), is allowed.

Most apps only need to set the category once, at launch. That said, you can change the category as often as you need to, and can do so whether your session is active or inactive. If your session is inactive, the category request is sent when you activate your session. If your session is already active, the category request is sent immediately.

Choosing the Best Category

The precise behaviors associated with each category are not under your app's control, but rather are set by the operating system. Apple may refine category behavior in future versions of iOS. Your best strategy is to pick the category that most accurately describes your intentions for the audio behavior you want. The appendix, Audio Session Categories and Modes (page 47), summarizes behavior details for each category.

To pick the best category, consider:

- Do you want to play audio, record audio, do both, or just perform offline audio processing?

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

- Is audio that you play essential or peripheral to using your app? If essential, the best category is one that supports continued playback when the Ring/Silent switch is set to silent. If peripheral, pick a category that goes silent with the Ring/Silent switch set to silent.
- Is other audio (such as the Music app) playing when the user launches your app? Checking during launch enables you to branch. For example, a game app could choose a category configuration that allows music to continue if it's already playing, or choose a different category configuration to support a built-in app soundtrack otherwise.

The following list describes the categories and the audio behavior associated with them. The `AVAudioSessionCategoryAmbient` category allows other audio to continue playing; that is, it is a mixable app. The remaining categories indicate that you want other audio to stop when your session becomes active. However, you can customize the non-mixing "playback" and "play and record" categories to allow mixing, as described in Fine-Tuning a Category (page 24).

- `AVAudioSessionCategoryAmbient`—Playback only. Plays sounds that add polish or interest but are not essential to the app's use. Using this category, your audio is silenced by the Ring/Silent switch and when the screen locks.
- `AVAudioSessionCategorySoloAmbient`—(Default) Playback only. Silences audio when the user switches the Ring/Silent switch to the "silent" position and when the screen locks. This category differs from the `AVAudioSessionCategoryAmbient` category only in that it interrupts other audio.
- `AVAudioSessionCategoryPlayback`—Playback only. Plays audio even with the screen locked and with the Ring/Silent switch set to silent. Use this category for an app whose audio playback is of primary importance.

> **Note:** If you choose an audio session category that allows audio to keep playing when the screen locks, you must set the `UIBackgroundModes` audio in your app's info.plist. See UIBackgroundModes for more information. You should normally *not* disable the system's sleep timer via the `idleTimerDisabled` property. If you do disable the sleep timer, be sure to reset this property to `NO` when your app does not need to prevent screen locking. The sleep timer ensures that the screen goes dark after a user-specified interval, saving battery power.

- `AVAudioSessionCategoryRecord`—Record only. Use `AVAudioSessionCategoryPlayAndRecord` if your app also plays audio.
- `AVAudioSessionCategoryPlayAndRecord`—Playback and record. The input and output need not occur simultaneously, but can if needed. Use for audio chat apps.
- `AVAudioSessionCategoryAudioProcessing`—Offline audio processing only. Performs offline audio processing and no playing or recording.

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

- `AVAudioSessionCategoryMultiRoute`—Playback and record. Allow simultaneous input and output for different audio streams, for example, USB and headphone output. A DJ app would benefit from using the multiroute category. A DJ often needs to listen to one track of music while another track is playing. Using the multiroute category, a DJ app can play future tracks through the headphones while the current track is played for the dancers.

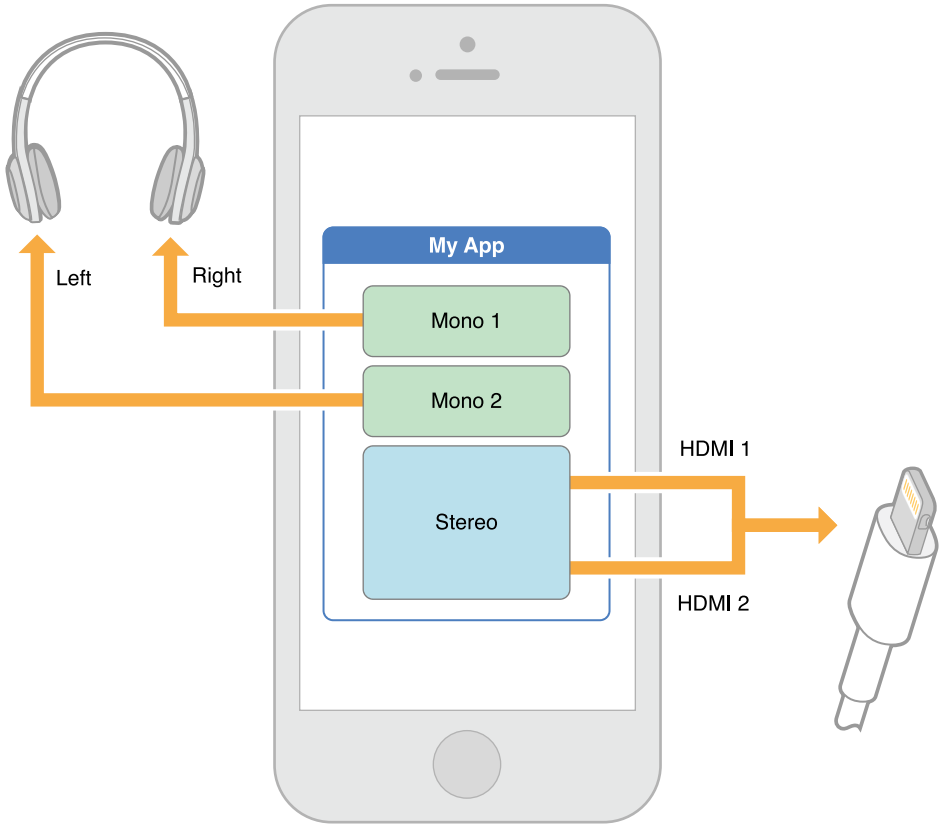
Expanding Options Using the Multiroute Category

The multiroute category works slightly differently than the other categories. All categories follow the "last in wins" rule, where the last device plugged into an input or output route is the dominant device. However, the multiroute category enables the app to use all of the connected output ports instead of only the last-in port. For example, if you are listening to audio through the HDMI output route and plug in a set of headphones, the audio plays through the headphones. Your app can continue playing audio through the HDMI output route while also playing audio through the headphones.

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

Your app can send different audio streams to different output routes. For example, your app could send one audio stream to the left headphone, another audio stream to the right headphone, and a third audio stream to the HDMI routes. Figure 2-1 shows an example of sending multiple files to different audio routes.

Figure 2-1 Sending different files to different audio routes



Depending on the device and any connected accessories, the following are valid output route combinations:

- USB and headphones
- HDMI and headphones
- LineOut and headphones

The multiroute category supports the use of a single input port.

Setting Your Audio Session Category

For most iOS apps, setting your audio session category at launch—and never changing it—works well. This provides the best user experience because the device's audio behavior remains consistent as your app runs.

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

To set the audio session category, call the `setCategory:error:` method as shown in Listing 2-1. For descriptions of all the categories, refer to Choosing the Best Category (page 18).

Listing 2-1 Setting the audio session category using the AV Foundation framework

```
NSError *setCategoryError = nil;
BOOL success = [[AVAudioSession sharedInstance]
                setCategory: AVAudioSessionCategoryAmbient
                      error: &setCategoryError];

if (!success) { /* handle the error in setCategoryError */ }
```

Using Modes to Specialize the Category

While categories set the base behaviors for your app, modes are used to specialize an audio session category. Set the mode for a category to further define the audio behaviors of your app. There are seven modes to choose from:

- `AVAudioSessionModeDefault`—Default mode that works with all categories and configures the device for general usage.
- `AVAudioSessionModeVoiceChat`—For Voice over IP (VoIP) apps. This mode can be used only with the `AVAudioSessionCategoryPlayAndRecord` category. Signals are optimized for voice through system-supplied signal processing, and the mode sets `AVAudioSessionCategoryOptionAllowBluetooth`.

  The set of allowable audio routes are optimized for voice chat experience. When the built-in microphones are used, the system automatically chooses the best combination of built-in microphones for the voice chat.
- `AVAudioSessionModeVideoChat`—For video chat apps such as FaceTime. The video chat mode can only be used with the `AVAudioSessionCategoryPlayAndRecord` category. Signals are optimized for voice through system-supplied signal processing and sets `AVAudioSessionCategoryOptionAllowBluetooth` and `AVAudioSessionCategoryOptionDefaultToSpeaker`.

  The set of allowable audio routes are optimized for video chat experience. When the built-in microphones are used, the system automatically chooses the best combination of built-in microphones for the video chat.

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

> **Note:** Apple recommends that apps using voice or video chat also use the Voice-Processing I/O audio unit. The Voice-Processing I/O unit provides several features for VOIP apps, including automatic gain correction, adjustment of voice-processing, and muting. See Voice-Processing I/O Unit for more information.

- `AVAudioSessionModeGameChat`—For game apps. This mode is set automatically by apps that use a `GKVoiceChat` object and the `AVAudioSessionCategoryPlayAndRecord` category. Game chat mode uses the same routing parameters as the video chat mode.
- `AVAudioSessionModeVideoRecording`—For apps that use the camera to capture video. The video recording mode can be used only with the `AVAudioSessionCategoryPlayAndRecord` and `AVAudioSessionCategoryRecord` categories. Signals are modified by system-supplied signal processing.

  Use the `AVCaptureSession` API in conjunction with the video recording mode for greater control of input and output routes. For example, setting the `automaticallyConfiguresApplicationAudioSession` property automatically chooses the best input route depending on the device and camera used.
- `AVAudioSessionModeMeasurement`—For apps that need to minimize the amount of system-supplied signal processing to input and output signals. This mode can only be used with the following categories: record, play-and-record, and playback. Input signals are routed through the primary microphone for the device.
- `AVAudioSessionModeMoviePlayback`—For apps that play movies. This mode can be used only with the `AVAudioSessionCategoryPlayback` category.

Choosing Categories and Modes for AirPlay

Only specific categories and modes support AirPlay. The following categories support both the mirrored and non-mirrored versions of Airplay:

- `AVAudioSessionCategorySoloAmbient`
- `AVAudioSessionCategoryAmbient`
- `AVAudioSessionCategoryPlayback`

The `AVAudioSessionCategoryPlayAndRecord` category only supports mirrored Airplay.

Modes only support AirPlay when used in conjunction with the play-and-record category. The following modes support AirPlay mirroring only:

- `AVAudioSessionModeDefault`

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

- `AVAudioSessionModeVideoChat`
- `AVAudioSessionModeGameChat`

Fine-Tuning a Category

You can fine-tune an audio session category in a variety of ways. Depending on the category, you can:

- Allow other audio (such as from the Music app) to mix with yours when a category normally disallows it
- Change the audio output route from the receiver to the speaker if no other route is available
- Specify that other audio should reduce in volume ("duck") when your audio session is active

You can override the interruption characteristic of the `AVAudioSessionCategoryPlayback`, `AVAudioSessionCategoryPlayAndRecord`, and `AVAudioSessionCategoryMultiRoute` categories so that other audio is allowed to mix with yours. To perform the override, apply the `AVAudioSessionCategoryOptionMixWithOthers` property to your audio session. When you set your app to be mixable, your app will not interrupt audio from other non-mixable apps when its audio session goes active. Also, your app's audio will not be interrupted by another app's non-mixable audio session, for example the Music app.

You can programmatically influence the audio output route. When using the `AVAudioSessionCategoryPlayAndRecord` category, audio normally goes to the receiver (the small speaker you hold to your ear when on a phone call). You can redirect audio to the speaker at the bottom of the phone by using the `overrideOutputAudioPort:error:` method.

Finally, you can enhance a category to automatically lower the volume of other audio when your audio is playing. This could be used, for example, in an exercise app. Say the user is exercising along to the Music app when your app wants to overlay a verbal message—for instance, "You've been rowing for 10 minutes." To ensure that the message from your app is intelligible, apply the `AVAudioSessionCategoryOptionDuckOthers` property to your audio session. When ducking takes place, all other audio on the device—apart from phone audio—lowers in volume. Apps that use ducking must manage their session's activation state. Activate the audio session prior to playing the audio and deactivate the session after playing the audio.

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

Recording Permission

Starting in iOS 7, your app must ask and receive permission from the user before you can record audio. If the user does not give your app permission to record audio, then only silence is recorded. The system automatically prompts the user for permission when you use a category that supports recording and the app attempts to use an input route.

Instead of waiting for the system to prompt the user for recording permission, you can also use the `requestRecordPermission:` method to manually ask the user for recording permission. Using the `requestRecordPermission:` method allows your app to get recording permission from the user at a time that doesn't interrupt the natural flow of the app. This provides a smoother experience for the user.

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

Responding to Interruptions

Adding audio session code to handle interruptions ensures that your app's audio continues behaving gracefully when a phone call arrives, a Clock or Calendar alarm sounds, or another app activates its audio session.

An *audio interruption* is the deactivation of your app's audio session—which immediately stops or pauses your audio, depending on which technology you are using. Interruptions happen when a competing audio session from an app activates and that session is not categorized by the system to mix with yours. After your session goes inactive, the system sends a "you were interrupted" message which you can respond to by saving state, updating the user interface, and so on.

Your app may get suspended following an interruption. This happens when a user decides to accept a phone call. If a user instead elects to ignore a call, or dismisses an alarm, the system issues an interruption-ended message, and your app continues running. For your audio to resume, your audio session must be reactivated.

Audio Interruption Handling Techniques

Handle audio interruptions by registering for the appropriate `NSNotification`. What you do within your interruption code depends on the audio technology you are using and on what you are using it for—playback, recording, audio format conversion, reading streamed audio packets, and so on. Generally speaking, you need to ensure the minimum possible disruption, and the most graceful possible recovery, from the perspective of the user.

Table 3-1 summarizes appropriate audio session behavior during an interruption. If you use the `AVAudioPlayer` or `AVAudioRecorder` objects, some of these steps are handled automatically by the system.

Table 3-1 What should happen during an audio session interruption

After interruption starts	• Save state and context • Update user interface
After interruption ends	• Restore state and context • Reactivate audio session if app appropriate • Update user interface

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

Table 3-2 summarizes how to handle audio interruptions according to technology. The rest of this chapter provides details.

Table 3-2 Audio interruption handling techniques according to audio technology

Audio technology	How interruptions work
AV Foundation framework	The `AVAudioPlayer` and `AVAudioRecorder` classes provide delegate methods for interruption start and end. Implement these methods to update your user interface and optionally, after interruption ends, to resume paused playback. The system automatically pauses playback or recording upon interruption and reactivates your audio session when you resume playback or recording. If you want to save and restore the playback position between app launches, save the playback position on interruption as well as on app quit.
Audio Queue Services, I/O audio unit	These technologies put your app in control of handling interruptions. You are responsible for saving playback or recording position and for reactivating your audio session after interruption ends.
OpenAL	When using OpenAL for playback, register for the appropriate `NSNotification` notifications—as when using Audio Queue Services. However, the delegate must additionally manage the OpenAL context. See OpenAL and Audio Interruptions (page 29).
System Sound Services	Sounds played using System Sound Services go silent when an interruption starts. They can automatically be used again if the interruption ends. Apps cannot influence the interruption behavior for sounds that use this playback technology.

Handling Interruptions From Siri

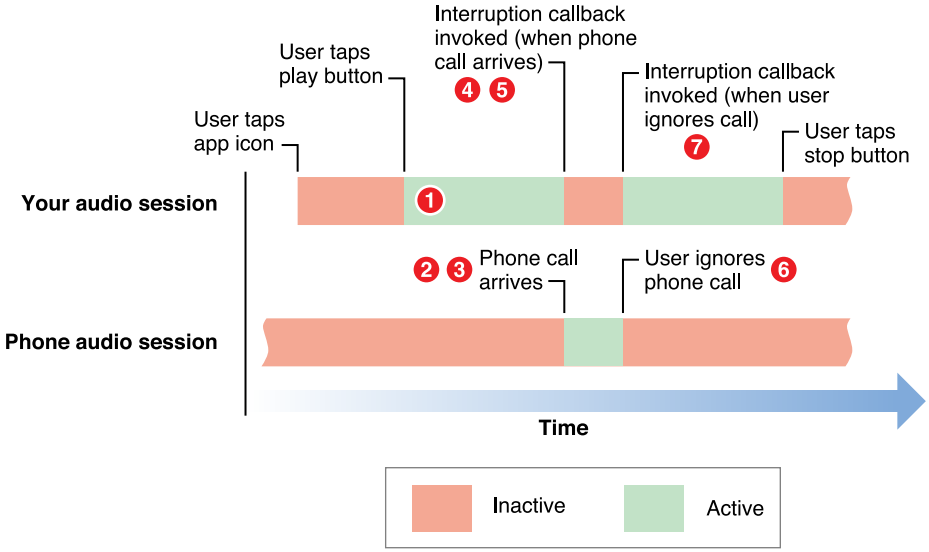
When your app is interrupted during playback by Siri, you must keep track of any remote control commands issued by Siri while the audio session is interrupted. During the interrupted state, keep track of any commands issued by Siri and respond accordingly when the interruption ends. For example, during the interruption, the user asks Siri to pause your app's audio playback. When your app is notified that the interruption has ended, it should not automatically resume playing. Instead, your app's UI should indicate that it is in a paused state.

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

The Interruption Life Cycle

Figure 3-1 illustrates the sequence of events before, during, and after an audio session interruption for a playback app.

Figure 3-1 An audio session gets interrupted



An interruption event—in this example, the arrival of a phone call—proceeds as follows. The numbered steps correspond to the numbers in the figure.

1. Your app is active, playing back audio.
2. A phone call arrives. The system activates the phone app's audio session.
3. The system deactivates your audio session. At this point, playback in your app has stopped.
4. The system invokes your interruption listener callback function indicating that your session has been deactivated.
5. Your callback takes appropriate action. For example, it could update the user interface and save the information needed to resume playback at the point where it stopped.
6. If the user dismisses the interruption—electing to ignore an incoming phone call in this case—the system invokes your callback method, indicating that the interruption has ended.
7. Your callback takes action appropriate to the end of an interruption. For example, it updates the user interface, reactivates your audio session, and resumes playback.
8. (Not shown in the figure.) If, instead of dismissing the interruption at step (6), the user accepted a phone call, your app is suspended. When the phone call ends, the end interruption is delivered.

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

There is no guarantee that a begin interruption will have an end interruption. Your app needs to be aware of switching to a foreground running state or the user pressing a play button. In either case, determine whether your app should reactivate its audio session.

OpenAL and Audio Interruptions

When using OpenAL for audio playback, implement an interruption listener callback function, as you do when using Audio Queue Services. However, your interruption code must additionally manage the OpenAL context. Upon interruption, set the OpenAL context to `NULL`. After the interruption ends, set the context to its prior state.

Using the AVAudioPlayer Class to Handle Audio Interruptions

The `AVAudioPlayer` class provides its own interruption delegate methods, described in *AVAudioPlayerDelegate Protocol Reference*. Likewise, the `AVAudioRecorder` class provides its own interruption delegate methods, described in *AVAudioRecorderDelegate Protocol Reference*. You use similar approaches for the two classes.

When your interruption-started delegate gets called, your audio player is already paused and your audio session is already deactivated. You can provide an implementation such as that shown in Listing 3-1.

Listing 3-1 An interruption-started delegate method for an audio player

```
- (void) audioPlayerBeginInterruption: (AVAudioPlayer *) player {
    if (playing) {
        playing = NO;
        interruptedOnPlayback = YES;
        [self updateUserInterface];
    }
}
```

After checking to ensure that the audio player was indeed playing when the interruption arrived, this method updates your app's state and then updates the user interface. (The system automatically pauses the audio player.)

If a user ignores a phone call, the system automatically reactivates your audio session and your interruption-ended delegate method gets invoked. Listing 3-2 shows a simple implementation of this method.

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

Listing 3-2 An interruption-ended delegate method for an audio player

```
- (void) audioPlayerEndInterruption: (AVAudioPlayer *) player {
    if (interruptedOnPlayback) {
        [player prepareToPlay];
        [player play];
        playing = YES;
        interruptedOnPlayback = NO;
    }
}
```

Responding to a Media Server Reset

The media server provides audio and other multimedia functionality through a shared server process. Although rare, it is possible for the media server to reset while your app is active. Register for the `AVAudioSessionMediaServicesWereResetNotification` notification to monitor for a media server reset. After receiving the notification, your app needs to do the following:

- Dispose of orphaned audio object and create new audio objects
- Reset any internal audio states being tracked, including all properties of `AVAudioSession`
- When appropriate, reactivate the `AVAudioSession` using the `setActive:error:` method

Important: Apps do not need to re-register for any `AVAudioSession` notifications or reset key-value-observers on `AVAudioSession` properties.

You can also register for the `AVAudioSessionMediaServicesWereLostNotification` notification if you want to know when the media server first becomes unavailable. However, most apps only need to respond to the reset notification. Only use the lost notification if the app must respond to user events that occur after the media server is lost, but before the media server is reset.

Providing Guidelines to the User

A user may not want an app to be interrupted by a competing audio session—for instance, when running an audio recorder to capture a presentation.

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

There is no programmatic way to ensure that an audio session is never interrupted. The reason is that iOS always gives priority to the phone. iOS also gives high priority to certain alarms and alerts—you wouldn't want to miss your flight now, would you?

The solution to guaranteeing an uninterrupted recording is for a user to deliberately silence their iOS device by taking the following steps:

1. In the Settings app, ensure that Airplane Mode turned on, for devices that have an Airplane mode.
2. In the Settings app, ensure that Do Not Disturb is turned on.
3. In the Calendar app, ensure that there are no event alarms enabled during the planned recording period.
4. In the Clock app, ensure that no clock alarms are enabled during the planned recording period.
5. For devices that have a Silent switch (called the *Ring/Silent switch* on iPhone), do not move the switch during the recording. When you change to Silent mode, an iPhone may vibrate, for example, depending on user settings.
6. Do not plug in or unplug a headset during recording. Likewise, do not dock or undock the device during recording.
7. Do not plug the device into a power source during the recording. When an iOS device gets plugged into power, it may beep or vibrate, according to the device and to user settings.

A user guide is a good place to communicate these steps to your users.

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

Optimizing Your App for Device Hardware

Using audio session properties, you can optimize your app's audio behavior for device hardware at runtime. This lets your code adapt to the characteristics of the device it's running on, as well as to changes made by the user (such as plugging in a headset or docking the device) as your app runs.

The audio session property mechanism lets you:

- Specify preferred hardware settings for sample rate and I/O buffer duration
- Query many hardware characteristics, among them input and output latency, input and output channel count, hardware sample rate, hardware volume setting, and whether audio input is available
- Respond to device specific notifications

The most commonly used property value change event is route changes, covered in Responding to Route Changes (page 37). You can also write callbacks to listen for changes in hardware output volume and changes in the availability of audio input.

Choosing Preferred Audio Hardware Values

Use the audio session APIs to specify preferred hardware sample rate and preferred hardware I/O buffer duration. Table 4-1 describes benefits and costs of these preferences.

Table 4-1 Choosing preferred hardware values

Setting	Preferred sample rate	Preferred I/O buffer duration
High value	*Example: 44.1 kHz* + High audio quality – Large file size	*Example: 500 mS* + Less-frequent disk access – Longer latency
Low value	*Example: 8 kHz* + Small file size – Low audio quality	*Example: 5 mS* + Low latency – Frequent disk access

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

For example, as shown in the top-middle cell of the table, you might specify a preference for a high sample rate if audio quality is very important in your app, and if large file size is not a significant issue.

The default audio I/O buffer duration (about 0.02 seconds for 44.1 kHz audio) provides sufficient responsiveness for most apps. A lower I/O duration can be set for latency-critical apps such as live musical instrument monitoring, but you won't need to change this value for most apps.

Setting Preferred Hardware Values

Set preferred hardware values prior to activating your audio session. When an app sets a preferred value, it does not take effect until the audio session is activated. Verify the selected values after your audio session has been reactivated. While your app is running, Apple recommends that you deactivate your audio session before changing any of the set values. Listing 4-1 shows how to set preferred hardware values and how to check the actual values being used.

Listing 4-1 Setting and querying hardware values

```
NSError *audioSessionError = nil;
AVAudioSession *session = [AVAudioSession sharedInstance];
[session setCategory:AVAudioSessionCategoryPlayback error:&audioSessionError];
if (audioSessionError) {
     NSLog(@"Error %ld, %@, (long)audioSessionError.code,
audioSessionError.localizedDescription);
}

NSTimeInterval bufferDuration =.005;
[session setPreferredIOBufferDuration:bufferDuration error:&audioSessionError];
if (audioSessionError) {
     NSLog(@"Error %ld, %@, (long)audioSessionError.code,
audioSessionError.localizedDescription);
}

double sampleRate = 44100.0
[session setPreferredSampleRate:samplerate error:&audioSessionError];
if (audioSessionError) {
     NSLog(@"Error %ld, %@, (long)audioSessionError.code,
audioSessionError.localizedDescription);
}
```

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

```
[[NSNotificationCenter defaultCenter] addObserver:self
                                          selector:@selector(handleRouteChange:)

name:AVAudioSessionRouteChangeNotification
                                            object:session];


[session setActive:YES error:&audioSessionError];
if (audioSessionError) {
     NSLog(@"Error %ld, %@, (long)audioSessionError.code,
audioSessionError.localizedDescription);
}


sampeRate = session.sampleRate;
bufferDuration = session.IOBuffferDuration;
NSLog(@"Sampe Rate:%0.0fHZ I/O Buffer Duration:%f", sampleRate, bufferDuration);
```

Querying Hardware Characteristics

Hardware characteristics of an iOS device can change while your app is running, and can differ from device to device. When you use the built-in microphone for an original iPhone, for example, recording sample rate is limited to 8 kHz; attaching a headset and using the headset microphone provides a higher sample rate. Newer iOS devices support higher hardware sample rates for the built-in microphone.

Your app's audio session can tell you about many hardware characteristics of a device. These characteristics can change at runtime. For instance, input sample rate may change when a user plugs in a headset. See *AVAudioSession Class Reference* for a complete list of properties.

Before you specify preferred hardware characteristics, ensure that the audio session is inactive. After establishing your preferences, activate the session and then query it to determine the actual characteristics. This final step is important because in some cases, the system cannot provide what you ask for.

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

Important: To obtain meaningful values for hardware characteristics, ensure that the audio session is initialized and active before you issue queries.

Two of the most useful audio session hardware properties are `sampleRate` and `outputLatency`. The `sampleRate` property contains the hardware sample rate of the device. The `outputLatency` property contains the playback latency of the device.

Specifying Preferred Hardware I/O Buffer Duration

Use the `AVAudioSession` class to specify preferred hardware sample rates and preferred hardware I/O buffer durations, as shown in Listing 4-2. To set preferred sample rate you'd use similar code.

Listing 4-2 Specifying preferred I/O buffer duration using the AVAudioSession class

```
NSError *setPreferenceError = nil;
NSTimeInterval preferredBufferDuration = 0.005;
[[AVAudiosession sharedInstance]
            setPreferredIOBufferDuration: preferredBufferDuration
                                   error: &setPreferenceError];
```

After establishing a hardware preference, always ask the hardware for the actual value, because the system may not be able to provide what you ask for.

Obtaining and Using the Hardware Sample Rate

As part of setting up for recording audio, you obtain the current audio hardware sample rate and apply it to your audio data format. Listing 4-3 shows how. You typically place this code in the implementation file for a recording class. Use similar code to obtain other hardware properties, including the input and output number of channels.

Before you query the audio session for current hardware characteristics, ensure that the session is active.

Listing 4-3 Obtaining the current audio hardware sample rate using the AVAudioSession class

```
double sampleRate;
sampleRate = [[AVAudioSession sharedInstance] currentHardwareSampleRate];
```

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

Running Your App in the Simulator

When you add audio session support to your app, you can run your app in the Simulator or on a device. However, the Simulator does not simulate audio session behavior and does not have access to the hardware features of a device. When running your app in the Simulator, you cannot:

- Invoke an interruption
- Change the setting of the Silent switch
- Simulate screen lock
- Simulate the plugging in or unplugging of a headset
- Query audio route information or test audio session category behavior
- Test audio mixing behavior—that is, playing your audio along with audio from another app (such as the Music app)

Because of the characteristics of the Simulator, you may want to conditionalize your code to allow partial testing in the Simulator.

One approach is to branch based on the return value of an API call. Ensure that you are checking and appropriately responding to the result codes from all of your audio session function calls; the result codes may indicate why your app works correctly on a device but fails in the Simulator.

In addition to correctly using audio session result codes, you can employ preprocessor conditional statements to hide certain code when it is running in the Simulator. Listing 4-4 shows how to do this.

Listing 4-4 Using preprocessor conditional statements

```
#if TARGET_IPHONE_SIMULATOR
#warning *** Simulator mode: audio session code works only on a device
    // Execute subset of code that works in the Simulator
#else
    // Execute device-only code as well as the other code
#endif
```

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

Responding to Route Changes

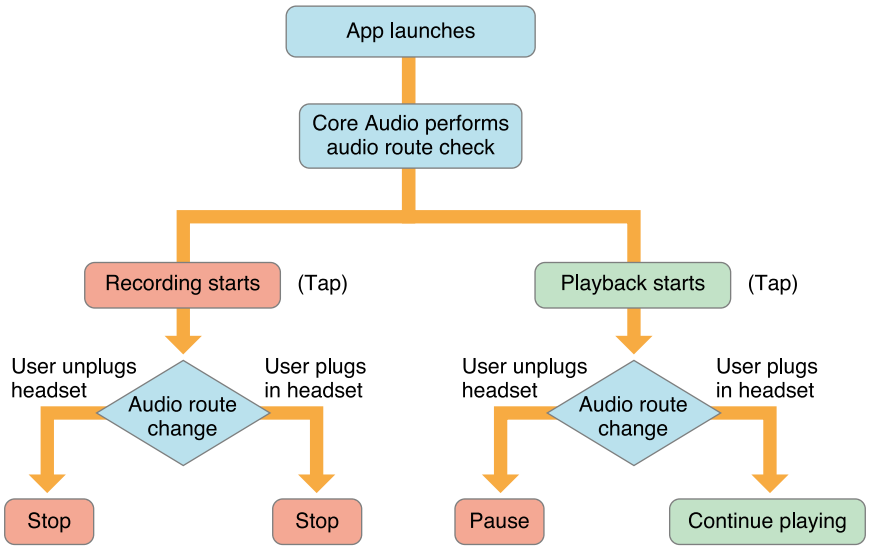
As your app runs, a user might plug in or unplug a headset, or use a docking station with audio connections. *iOS Human Interface Guidelines* describes how iOS apps should respond to such events. To implement the recommendations, write audio session code to handle audio hardware route changes. Certain types of apps, like games, don't always have to respond to route changes. However other types of apps, such as media players, must respond to all route changes.

Varieties of Audio Hardware Route Change

An *audio hardware route* is a wired electronic pathway for audio signals. When a user of an iOS device plugs in or unplugs a headset, the system automatically changes the audio hardware route. Your app can listen for such changes by way of `AVAudioSessionRouteChangeNotification`.

Figure 5-1 depicts the sequence of events for various route changes during recording and playback. The four possible outcomes, shown across the bottom of the figure, result from actions taken by a property listener callback function that you write.

Figure 5-1 Handling audio hardware route changes



2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

In the figure, the system initially determines the audio route after your app launches. It continues to monitor the active route as your app runs. Consider first the case of a user tapping a Record button in your app, represented by the "Recording starts" box on the left side of the figure.

During recording, the user may plug in or unplug a headset—see the diamond-shaped decision element toward the lower-left of the figure. In response, the system sends `AVAudioSessionRouteChangeNotification` containing the reason for the change and the previous route. Your app should stop recording.

The case for playback is similar but has different outcomes, as shown on the right of the figure. If a user unplugs the headset during playback, your app should pause the audio. If a user plugs in the headset during playback, your app should simply allow playback to continue.

The *AddMusic* sample code project demonstrates how to implement the playback portion of this behavior.

Responding to Audio Hardware Route Changes

There are two parts to configuring your app to respond to route changes:

1. Implement methods to be invoked upon a route change.
2. Register for the `AVAudioSessionRouteChangeNotification` notification to respond to route changes.

For example, your app receives a notification when a user unplugs the headset during playback. Following Apple guidelines, your app pauses. Your app can then provide a display that prompts the user to continue playing.

When the system sends a route-change notification, it provides the information you need to figure out which action to take. Register for the `AVAudioSessionRouteChangeNotification` notification as shown:

```
NSNotificationCenter *nc [NSNotificationCenter defaultCenter];
[nc addObserver:self
       selector:routeChanged:
           name:AVAudioSessionRouteChangeNotification
         object:nil];
```

After the notification is received, your app calls the method you designate and changes its behavior based on the information contained by the notification. The `AVAudioSessionRouteChangeNotification` contains a `userInfo` dictionary that describes:

- Why the route changed

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

- What the previous route was

The keys for the dictionary are `AVAudioSessionRouteChangeReasonKey` and `AVAudioSessionRouteChangePreviousRouteKey`. The various reasons why a hardware audio route might have changed—accessed by the `AVAudioSessionRouteChangeReasonKey` key—are listed and described in `AVAudioSessionRouteChangeReason`. The previous route information—accessed by the `AVAudioSessionRouteChangePreviousRouteKey` key—is an object that describes the audio route setting from before the route change.

One reason for the audio hardware route change in iOS is `AVAudioSessionRouteChangeReasonCategoryChange`. In other words, a change in audio session category is considered by the system—in this context—to be a route change, and will invoke a route change notification. As a consequence, if your notification response method is intended to respond only to headset plugging and unplugging, it should explicitly ignore this type of route change.

Here's a practical example. A well-written recorder/playback app sets the audio session category when beginning playback or recording. Because of this, a route-change notification is sent upon starting playback (if you were previously recording) or upon starting recording (if you were previously playing back). Clearly, such an app should not pause or stop each time a user taps Record or Play. To avoid inappropriate pausing or stopping, the method called upon receipt of the notification should branch based on the reason for the route change and simply return if it was a category change. In this case, the category change is from the system's perspective and not the app's perspective.

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

Fine-Tuning an Audio Session for Players

Objective-CSwift

Movie players let you play movies from a file or a network stream. Music players let you play audio content from a user's music library. To use these objects in coordination with your app audio, consider their audio session characteristics.

- Music players (instances of the `MPMusicPlayerController` class) always use a system-supplied audio session.
- Movie players (instances of the `MPMoviePlayerController` class) use your app's audio session by default, but can be configured to use a system-supplied audio session.

Working with Music Players

To play audio from a user's music library along with your own sounds (as described in *iPod Library Access Programming Guide*), you use the `AVAudioSessionCategoryAmbient` category, a so-called *mixable* category configuration for your audio session, or the `AVAudioSessionCategoryOptionMixWithOthers` option with a compatible category. Using this category ensures that your sounds will not interrupt a music player—nor will a music player's sounds interrupt yours.

> **Important:** Do not attempt to use a music player without configuring a mixable category for your audio session.

The system automatically handles route changes and interruptions for music players. You cannot influence this built-in behavior. As long as you correctly manage your app's audio session as described here and in previous chapters, you can rely on a music player to take care of itself, as a user plugs in a headset, an alarm sounds, or a phone call arrives.

You can configure your audio session so that sound from a music player ducks (lowers in volume) when audio from your app plays. For details on ducking and how to enable it, see Fine-Tuning a Category (page 24).

For a description of the music player class, see *MPMusicPlayerController Class Reference*.

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

Working with Movie Players

By default, a movie player shares your app's audio session. In effect, a movie player transcends the notion of mixing with your app's audio; the movie player's audio behaves as though it belongs to your app. No matter which playback category you choose, and no matter how you configure that category, your audio and the movie player's audio never interrupt each other.

Sharing your audio session also gives you control over how a movie interacts with audio from other apps, such as the Music app. For example, if you set your category to `AVAudioSessionCategoryAmbient` and share your session, Music audio is not interrupted when a movie starts in your app. Sharing your audio session also lets you specify whether or not movie audio obeys the Ring/Silent switch.

To configure audio behavior for a movie, determine the behavior you want and then perform appropriate audio session configuration, as described in Table 6-1. For details on setting up your audio session, see Defining an Audio Session (page 10).

Table 6-1 Configuring audio sessions when using a movie player

Desired behavior	Audio session configuration
Playing a movie silences all other audio	• If your app does not play audio, do not configure an audio session. • If your app does play audio, configure an audio session and set its mix-ability according to whether or not you want to mix with the Music app and other audio. • In either case, tell the movie player to use its own audio session: `myMoviePlayer.useappAudioSession = NO`
Movie and app audio mix, but other audio, including the Music app, is silenced	• Configure an audio session using a *non-mixable* category. • Take advantage of the movie player's default `useApplicationAudioSession` value of `YES`.
All audio mixes	• Configure an audio session using a *mixable* category configuration. • Take advantage of the movie player's default `useApplicationAudioSession` value of `YES`.

Manage your app's audio session as usual in terms of route changes and interruptions, as described in Responding to Route Changes (page 37) and Responding to Interruptions (page 26). Enable ducking, if desired, as described in Fine-Tuning a Category (page 24).

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

If you have configured a movie player to use its own audio session, there's some cleanup to perform. To restore the movie player's ability to play audio after a movie finishes, or the user dismisses it:

1. Dispose of the movie player—even if you intend to play the same movie again later.
2. Reactivate your audio session.

For a description of the movie player class, see *MPMoviePlayerController Class Reference*.

Using the Media Player Framework Exclusively

If your app is using a movie player only, or a music player only—and you are not playing your own sounds—then you should not configure an audio session.

If you are using a movie player exclusively, you must tell it to use its own audio session, as follows:

```
myMoviePlayer.useappAudioSession = NO
```

If you are using a movie player *and* a music player, then you probably want to configure how the two interact; for this, you must configure an audio session, even though you are not playing app audio per se. Use the guidance in Table 6-1 (page 41).

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

Audio Guidelines By App Type

The latest driving game does not have the same audio requirements as a real-time video chat app. The following sections provide design guidelines for different types of audio apps.

Audio Guidelines for Game Apps

Most games require user interaction for anything to happen in the game. Use the `AVAudioSessionCategoryAmbient` or `AVAudioSessionCategorySoloAmbient` categories when designing games. When users bring up another app or lock the screen, they do not expect the app to continue playing. Often the user wants the audio from another app to continue playing while the game app plays.

Apple recommends the following guidelines:

- Play app sound effects while allowing another app's audio to play.
- Play app soundtrack audio when other audio is not playing, otherwise allow the previous audio to play.
- Always attempt to reactivate and resume playback after an end interruption event.
- Ignore all route changes unless the app specifically needs to pay attention to them.
- Set the audio category before displaying a video splash on app launch.

Audio Guidelines for User-Controlled Playback and Recording Apps

Video recording apps and apps such as Pandora and Netflix have the same guidelines. These types of apps use the `AVAudioSessionCategoryRecord`, `AVAudioSessionCategoryPlayAndRecord`, or `AVAudioSessionCategoryPlayback` categories and allow other apps to mix with them. These types of apps typically do not duck the audio of other apps. The UI will include a play/pause button or a record/pause button.

Apple recommends the following guidelines:

- Wait for the user to press the play/record button when the app enters the foreground before activating the audio session.
- Keep the audio session active throughout the app's lifetime unless it is interrupted.

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

- Update the UI to indicate that audio has paused when it is interrupted. Do not deactivate the audio session or pause/stop player objects.
- Check for the presence of the `AVAudioSessionInterruptionOptionKey` constant and honor its value after an end interruption. Don't start playing audio again unless the app was playing prior to the interruption.
- Pause the audio session due to a route change caused by an unplug event, but keep the audio session active.
- Assume the app's audio session is inactive when it transitions from a suspended to foreground state. Reactive the audio session when the user presses the play button.
- Ensure that the audio `UIBackgroundModes` flag is set.
- Use the `AVAudioSessionCategoryPlayAndRecord` category instead of the `AVAudioSessionCategoryRecord` category.
- Use a background task instead of streaming silence to keep the app from being suspended.
- Use a `MPVolumeView` object for the volume slide and route picker.
- Ask the user for permission to record input using the `requestRecordPermission:` method. Don't rely on the iOS to prompt the user.
- Register for remote control events.

Audio Guidelines for VoIP and Chat Apps

VoIP and chat apps require that both input and output routes are available. These types of apps use the `AVAudioSessionCategoryPlayAndRecord` category and do not mix with other apps.

Apple recommends the following guidelines:

- Only activate an audio session when the user answers or initiates a call.
- Update the UI to reflect the call's audio has been interrupted after an interruption notification.
- Do not activate an audio session after an interruption until the user answers or initiates a call.
- Deactivate the audio session after a call ends using the `AVAudioSessionSetActiveOptionNotifyOthersOnDeactivation` constant.
- Ignore all route changes unless the app specifically needs to pay attention to them.
- Ensure that the audio `UIBackgroundModes` flag is set.
- Use a `MPVolumeView` object for the volume slide and route picker.

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

- Ask the user for permission to record input using the `requestRecordPermission:` method. Don't rely on the iOS to prompt the user.

Audio Guidelines for Metering Apps

Metering apps want the minimal amount of system-supplied signal processing applied to the input and output routes. Set the `AVAudioSessionCategoryPlayAndRecord` category and the measurement mode to minimize signal processing. Also, apps of this type will not mix with other apps.

Apple recommends the following guidelines:

- Always attempt to reactivate and resume playback after an end interruption event.
- Ignore all route changes unless the app specifically needs to pay attention to them.
- Set the audio category before displaying a video splash on app launch.
- Ask the user for permission to record input using the `requestRecordPermission:` method. Don't rely on the iOS to prompt the user.

Audio Guidelines for Browser-like Apps That Sometimes Play Audio

Apps like Facebook and Instagram don't record audio, only playback audio and video. They use the `AVAudioSessionCategoryPlayback` category and do not obey the ringer switch. These apps also do not mix with other apps.

Apple recommends the following guidelines:

- Always wait for the user to initiate playback.
- Deactivate the audio session after video ends and set the `AVAudioSessionSetActiveFlags_NotifyOthersOnDeactivation` key.
- Pause the audio session due to a route change caused by an unplug event, but keep the audio session active.
- Register for remote control events while video is playing and unregister when the video ends.
- Update the UI when the app receives a begin interruption event.
- Wait for the user to initiate playback after receiving an end interruption event.

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

Audio Guidelines for Navigation and Workout Apps

Navigations and workout apps use the `AVAudioSessionCategoryPlayback` or `AVAudioSessionCategoryPlayAndRecord` categories. The audio from these apps are typically short prompts and will mix with other apps. As it is assumed that the user wants to hear the audio from these apps even when other apps are playing, these apps will duck audio from other apps.

Apple recommends the following guidelines:

- Do not activate the audio session until a prompt needs to be played.
- Always deactivate the audio session after a prompt is played.
- Ignore all interruptions and route changes.

Audio Guidelines for Cooperative Music Apps

Cooperative music apps are designed to play while other apps are playing. These types of apps will use the `AVAudioSessionCategoryPlayback` or `AVAudioSessionCategoryPlayAndRecord` category and will mix with other apps.

Apple recommends the following guidelines:

- If the app does not have a start/stop button, then also follow the guidelines for game apps.
- If the UI contains a start/stop button, only activate the audio session when the user presses the play button.
- Do not sign up for remote control events.
- Ensure that the audio `UIBackgroundModes` flag is set.
- If the app records user input, ask the user for permission to record input using the `requestRecordPermission:` method. Don't rely on the iOS to prompt the user.

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

Audio Session Categories and Modes

You tell iOS your app's audio intentions by designating a category for your audio session. Table B-1 provides details on each of the categories. The default category, `AVAudioSessionCategorySoloAmbient`, is shaded. For an explanation of how categories work, see Working with Categories (page 18). For an explanation of the mixing override switch, see Choosing the Best Category (page 18).

Table B-1 Audio session category behavior

Category identifiers	Silenced by the Ring/Silent switch and by screen locking [see note]	Interrupts non-mixable apps audio	Allows audio input (recording) and output (playback)
`AVAudioSessionCategoryAmbient`	Yes	No	Output only
`AVAudioSessionCategoryAudio-Processing`	–	Yes	No input and no output
`AVAudioSessionCategoryMultiRoute`	No	Yes	Input and output
`AVAudioSessionCategoryPlayAnd-Record`	No	Yes by default; no by using override switch	Input and output
`AVAudioSessionCategoryPlayback`	No	Yes by default; no by using override switch	Output only

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

Category identifiers	Silenced by the Ring/Silent switch and by screen locking [see note]	Interrupts non-mixable apps audio	Allows audio input (recording) and output (playback)
`AVAudioSessionCategoryRecord`	No (recording continues with the screen locked)	Yes	Input only
`AVAudioSessionCategorySoloAmbient`	Yes	Yes	Output only

Note: For your app to continue playing audio when the Ring/Silent switch is set to silent and the screen is locked, make sure the `UIBackgroundModes audio` key has been added to your app's `Info.plist` file. This requirement is in addition to your using the correct category.

Table B-2 provides a list of modes and what categories each mode can be used with.

Table B-2 Modes and associated categories

Mode identifiers	Compatible categories
`AVAudioSessionModeDefault`	All
`AVAudioSessionModeVoiceChat`	`AVAudioSessionCategoryPlayAndRecord`
`AVAudioSessionModeGameChat`	`AVAudioSessionCategoryPlayAndRecord`
`AVAudioSessionModeVideoRecording`	`AVAudioSessionCategoryPlayAndRecord` `AVAudioSessionCategoryRecord`
`AVAudioSessionModeMoviePlayback`	AVAudioSessionCategoryPlayback

`AVAudioSessionModeMeasurement`	`AVAudioSessionCategoryPlayAndRecord`
	`AVAudioSessionCategoryRecord`
	AVAudioSessionCategoryPlayback

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

Mode identifiers	Compatible categories
AVAudioSessionModeVideoChat	AVAudioSessionCategoryPlayAndRecord

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

Document Revision History

This table describes the changes to *Audio Session Programming Guide*.

Date	Notes
2014-09-17	Added information on AVAudioSessionCategoryMultiRoute able to be modified by AVAudioSessionCategoryOptionMixWithOthers.
2014-03-10	Updated document for iOS 7. Removed all C references.
2012-12-13	Added information on how to handle interruptions when using an audio processing graph.
2010-11-15	Added audio session considerations for the playInputClick method.
2010-09-01	Updated to explain how to support application lifecycle in iOS 4.
2010-07-09	Minor changes.
2010-04-12	Updated for iOS 3.2 by describing changes in the behavior of movie player audio sessions.
2010-01-29	TBD
2010-01-20	Added information on using hardware preferences and on handling screen locking.
2009-10-19	TBD
2009-09-09	Updated for iOS 3.1 to include descriptions of new audio session properties.
2008-11-13	New document for iOS 2.2 that explains how to use an iOS application's audio session.

2014-09-17 | Copyright © 2014 Apple Inc. All Rights Reserved.

Apple Inc.
Copyright © 2014 Apple Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer or device for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-branded products.

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Apple, the Apple logo, AirPlay, Cocoa, Cocoa Touch, FaceTime, iPhone, iPod, Objective-C, Safari, and Siri are trademarks of Apple Inc., registered in the U.S. and other countries.

IOS is a trademark or registered trademark of Cisco in the U.S. and other countries and is used under license.

APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT, ERROR OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

Some jurisdictions do not allow the exclusion of implied warranties or liability, so the above exclusion may not apply to you.